Classifying Android Malware through Subgraph Mining

Fabio Martinelli, Andrea Saracino, Daniele Sgandurra

Security Group @ IIT-CNR, Pisa, Italy
daniele.sgandurra@iit.cnr.it

SETOP 2013
12/09/2013

Outline

Part I

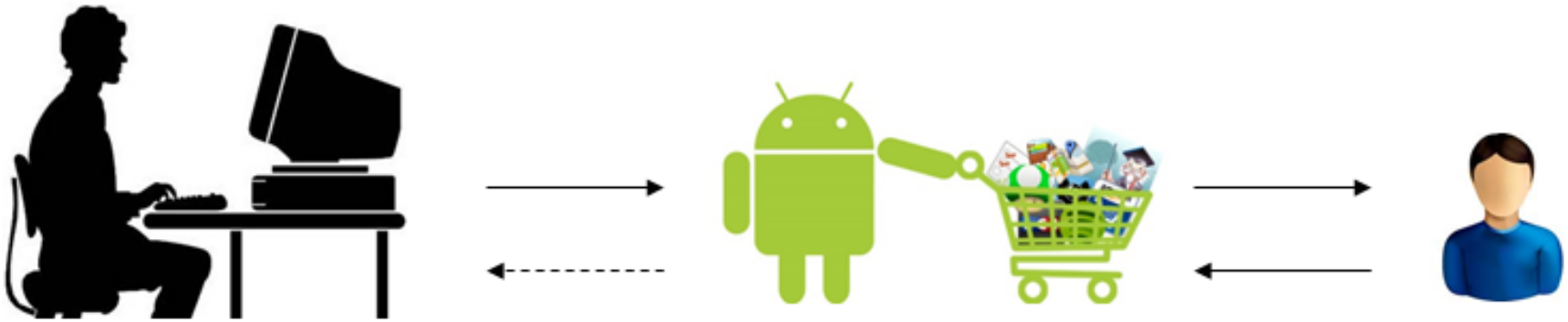
Mobile Security

Apps

- Current operative systems for mobile devices are based upon the concept of apps: new paradigm for application distribution.
- Apps are lightweight applications that are distributed through on-line marketplaces, such as the Apple AppStore or Android Google Play.

Apps

- Users browse apps and install them directly on their devices.
- This model is affected by some security and trust issues:
  - some apps are produced by third-party developers, which can be malicious ones,
  - some markets do not ensure the security and quality of the published apps.

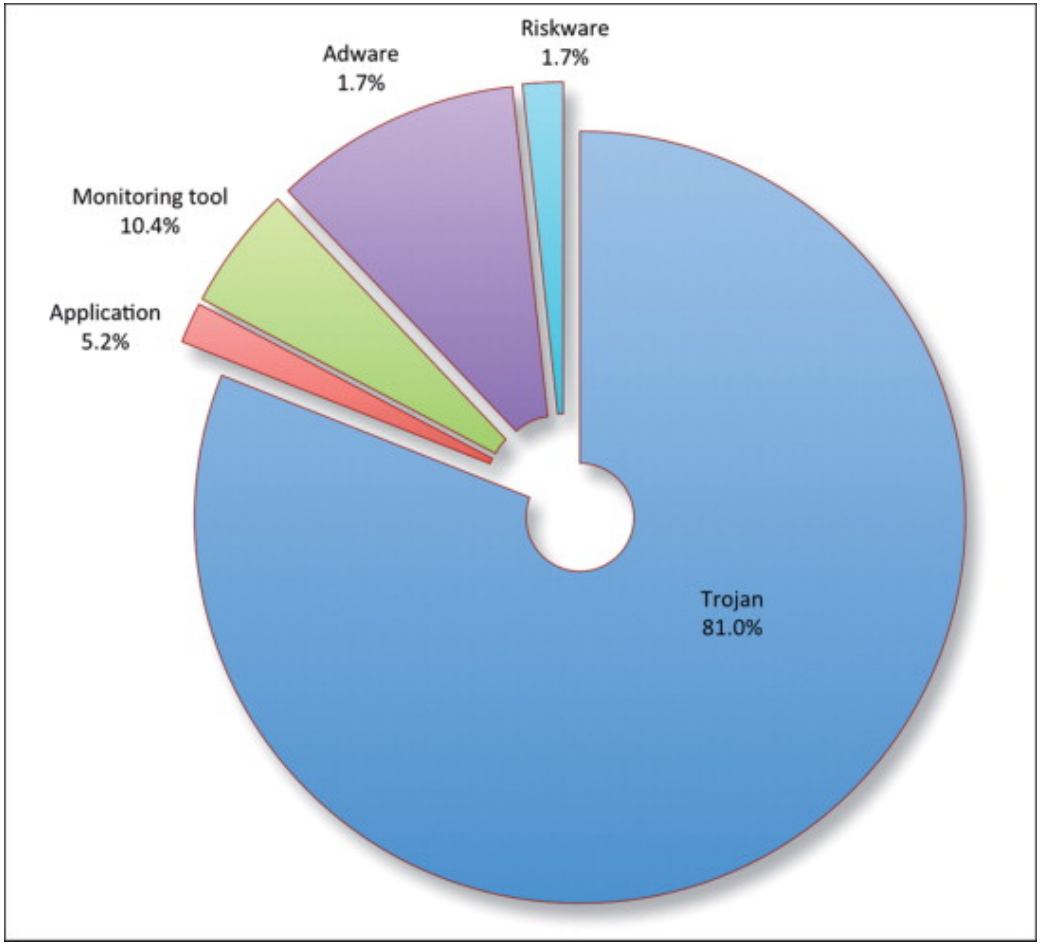


Malware

- A large number of malware has been found hidden in apps distributed in the Android markets
- Some apps are Trojanized apps, i.e. apps that look and work as genuine applications, but they hide inside new code that misbehaves in the background:
  - Trojanized apps form more than 80% of the total malware.

Mobile Threats by Type

Source: F-Secure.



Some Examples of Malware

- SMS Trojans: maliciously send stealthy SMS messages to leak the user credit, subscribe the user to premium services.
- Private Data Trojans: maliciously retrieve private data like contact lists, IMEI and IMSI codes, received/sent SMS messages, and send user's private data to the attacker.
- Rootkits: malware that obtain root privileges, exploiting a system weakness, then open a backdoor for the attacker. Installing other malicious applications, acting as a C&C bot.

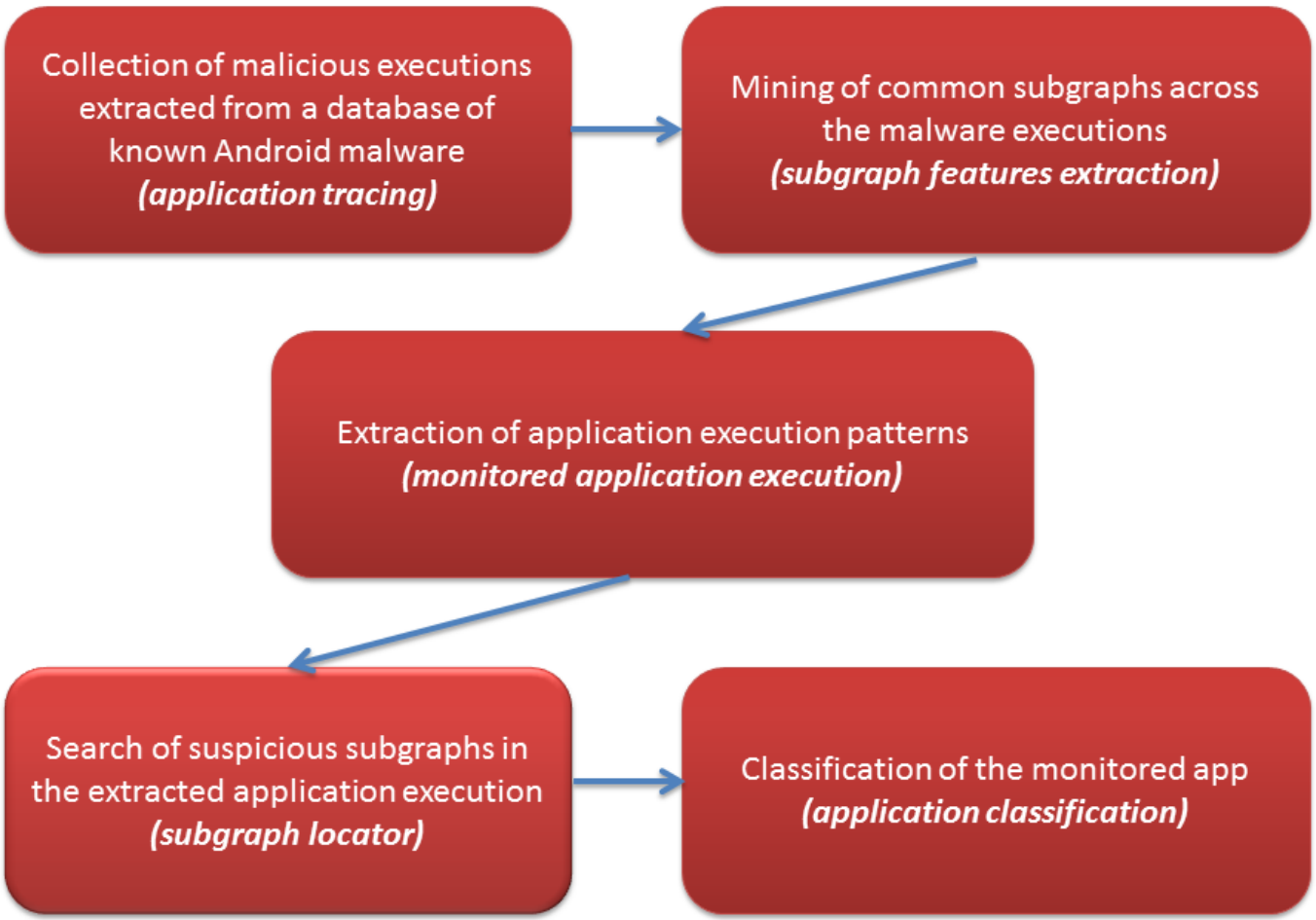
Part II

CAMAS

CAMAS

- CAMAS is a framework for Classification of Android MAlware through Subgraphs.
- It analyzes malware executions at the lowest level.
- It mines for common execution subgraphs, by looking for subgraphs that are frequent among the analyzed malware.
- CAMAS analyzes apps that are downloaded from marketplaces and it monitors their executions.
- If the analyzed executions contain suspicious subgraphs, the application is considered malicious.

Classification Process



Malware Monitoring and Graph Creation

1. Malware monitoring
   - to build a representative set of malware behaviors, CAMAS needs to find actions that are frequently performed by malware:
     - these actions are represented by subgraphs of system calls;
2. Graph creation, using cluster of related system calls:
   - to extract the common pattern of malware behavior;
   - common subgraph mining across malware, by carefully choosing the frequency, nodes and edges number.

Application Trace

- In CAMAS, an application trace is a sequence of system calls.
- CAMAS expresses the trace through an oriented graph $G = (V, E)$, where each system call is a node $v_i \in V$, and edges $e_{i,j} \in E$ represent the transition from a system call v_i into the next one v_j.
- In this graph, each node is a system call and each edge is labeled with a sequence number that represents the position in the trace.
- The resulting graph may contain more occurrences of the same edge (multi-graph).

GraphCreator

- A Linux kernel-module and an Android application that reads a shared buffer are included in the GraphCreator CAMAS component.
- The concept of ActionNodes is exploited by the GraphCreator to build execution graphs.
- To this end, the trace of system calls issued by the tested malware is converted into a multi-graph of ActionNodes:
  - more expressive than a graph of simple system calls.
- An ActionNode is composed by the automaton of the system calls performed consecutively on the same file.

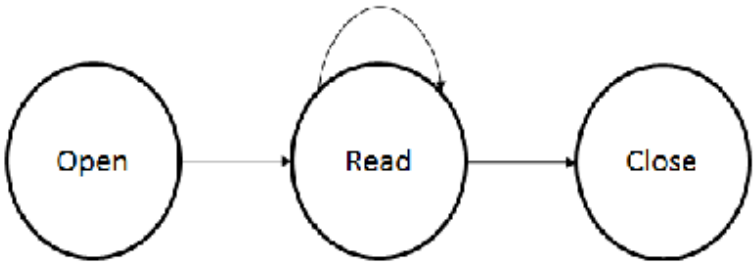
ActionNode

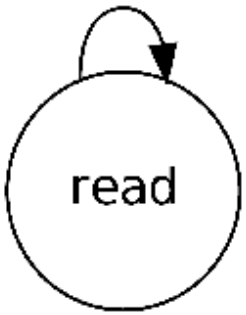
- CAMAS exploits the notion of ActionNode:
  - cluster of related system calls: sub-graph of system calls that are consecutively issued in the trace and that are bound by some relation;
  - form an action (a high-level operation).
- The relation should produce a meaningful action:
  - an ActionNode is composed by the automaton of the system calls performed consecutively on the same file descriptor.

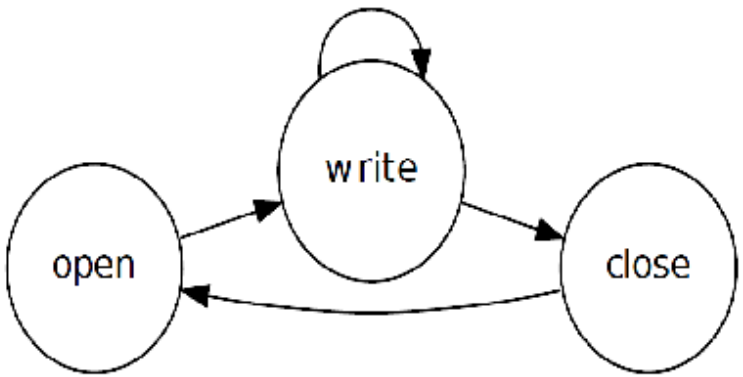
ActionNode

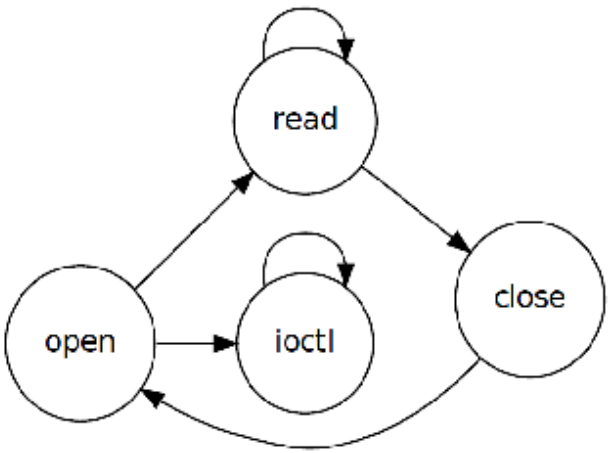
- An application trace can be seen as a graph whose nodes are actions.
- The framework can define a larger number of nodes, hence more detailed signatures.
- The nodes that compose an ActionNode, i.e. system calls, are called SysCallNodes.
- CAMAS considers as SysCallNodes system calls that act on files, namely the `open`, `read`, `write`, `close`, `ioctl`.

ActionNode Examples









Subgraph Mining

- After having generated the multi-graphs of ActionNodes from malware executions, CAMAS exploits a mining algorithm to find common execution subgraphs from these multi-graphs.
- Set of frequent subgraphs with an increasing number of ActionNodes are extracted from the traces using the ParSeMiS toolset.
- The ParSeMiS toolset returns the set of subgraphs with the same structure, which can be found in at least a chosen (high) percentage of malicious different malware executions.

Subgraph Mining

- The most important parameter: frequency of these subgraphs:
  - percentage of malware having the same subgraph in their multi-graphs of ActionNodes;
  - also subgraphs that are frequent in any kind of apps (i.e., even good ones) could be found;
  - refinement process, to find those features that are more representative (number of common subgraphs mined can be extremely high).
- Other parameters: minimum edge and the minimum node number contained in each mined subgraphs.

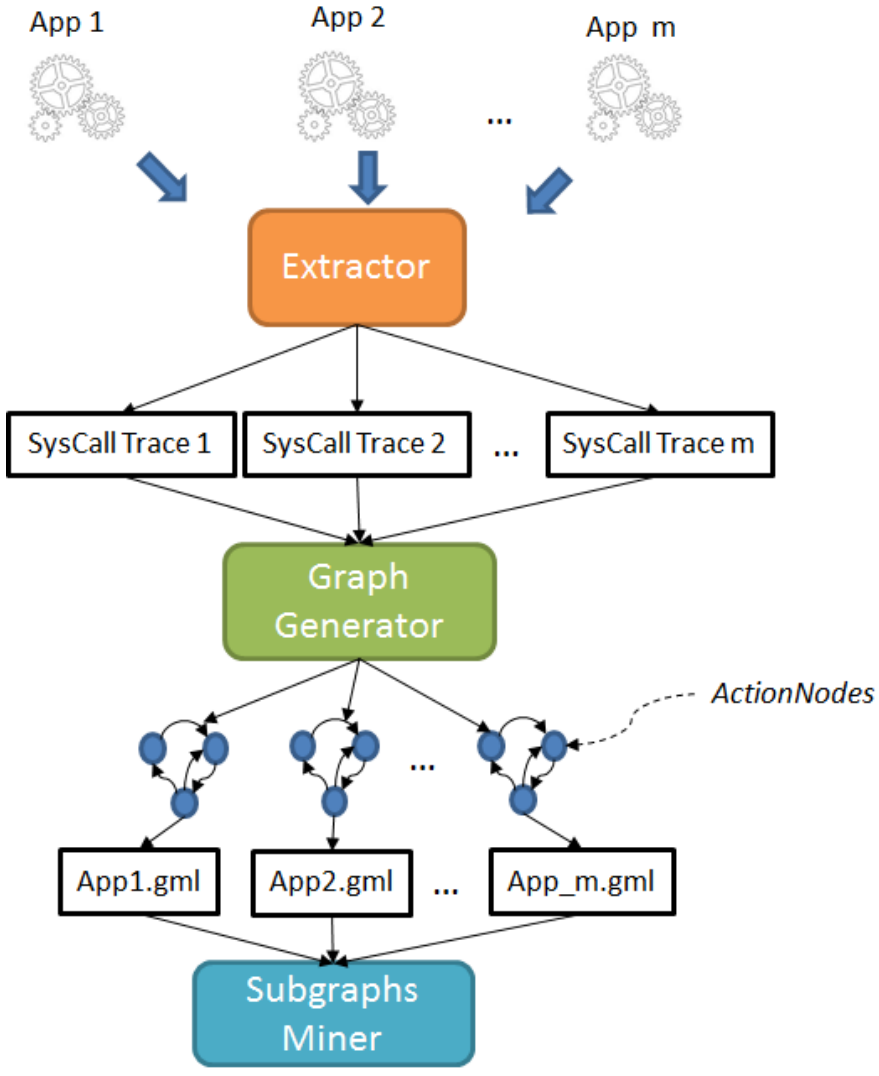
Algorithm to Mine Frequent Subgraphs (SubGraphMiner)

1. Select subgraphs that exist in at least 50% of the analyzed malware.
2. The minimum number of nodes is gradually increased:
   - starting from a number of nodes that is set to an acceptable size so as not to produce too many subgraphs,
   - increased until common subgraphs are not found anymore.
3. Increase the percentage of analyzed malware required to contain a mined subgraph and go back to step 2.
4. Save common subgraphs in the database in a GML format.

Simplified Algorithm to Mine Frequent Subgraphs

```
Set frequency at 50% of the analyzed number of malware;
Set node number at an acceptable size after a refinement process;
while frequency ≤ 100% do
    mine for commons subgraphs;
    if there are still subgraphs then
        add found subgraphs to frequency-node-edges.GML file;
        increase nodes number;
    else
        increase frequency of 1 step;
        set node number at an acceptable size after a refinement
        process;
    end
end
```

First Part of The Workflow



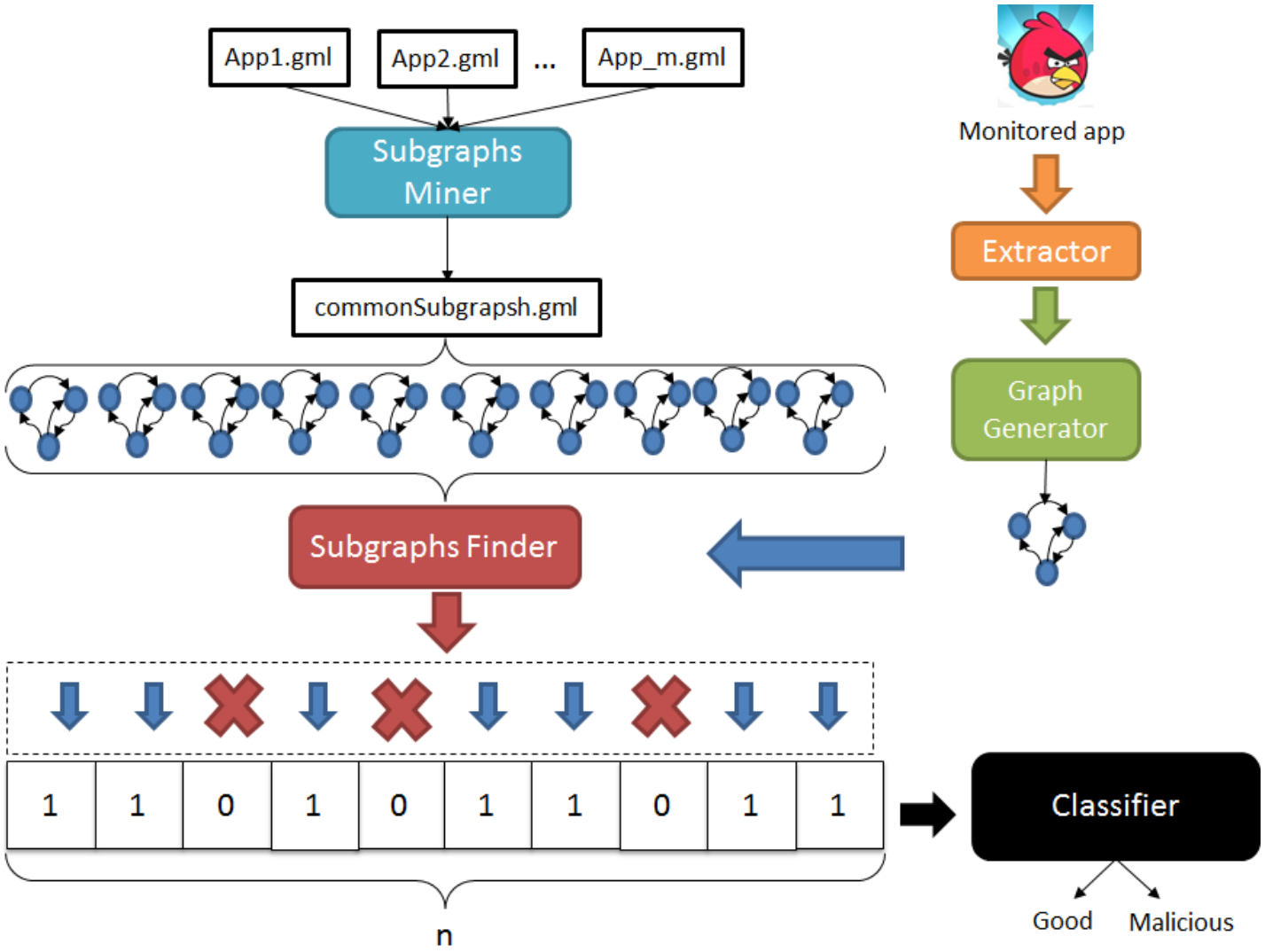
Classification

- Once the malicious subgraphs database has been created, CAMAS analyzes the behaviors of unknown apps.
- A monitored app is monitored for a sufficiently long time interval simulating at best real use-cases.
- A multi-graph of ActionNodes is built from the app execution traces.

SubgraphFinder

- SubgraphFinder discovers whether the ActionNodes multi-graph of a monitored application includes some malicious subgraphs.
- For each malicious subgraph MS_i in the database, SubgraphFinder checks if MS_i exists in the monitored application multi-graph.
- SubgraphFinder generates a binary vector of n elements, where n is the number of malicious subgraphs in the database of malware.
- Each element i of this vector is set to 1 if the subgraph MS_i is found in the monitored application, 0 otherwise.
- At the end, this vector is given to a classifier to test whether the application is considered a malware or not.

Second Part of The Workflow



Part III

Results

Implementation

- The whole CAMAS framework has been implemented in Java, except for the tracing kernel module (written in C):
  - 3,000 LOC.
- The testbed of the experiments was:
  - Android emulator with modified kernel (2.6.29), which logs the trace system calls on text files;
  - CAMAS is run on an external server, which receives log files and runs:
    - SubGraphMiner (during the training phase)
    - SubGraphFinder (during the monitoring phase).

Tested Malware

Collected representative malicious subgraphs from 12 real malware:

- KMIN
  - com.km.launcher
- ROOT EXPLOIT
  - com.z4mod.z4root:three
  - Super.mobi.eraser
  - com.z4mod.z4root
  - com.zft
  - com.itfunz.itfunzsupertools
  - com.droiddream.android.afdvancedfm
  - com.aps.hainguyen273.app2card
- SMS TROJAN
  - tp5x.WGt12
  - com.software.installer
- MOGAVA
  - ir.sharif.iranianfoods

Creating GML

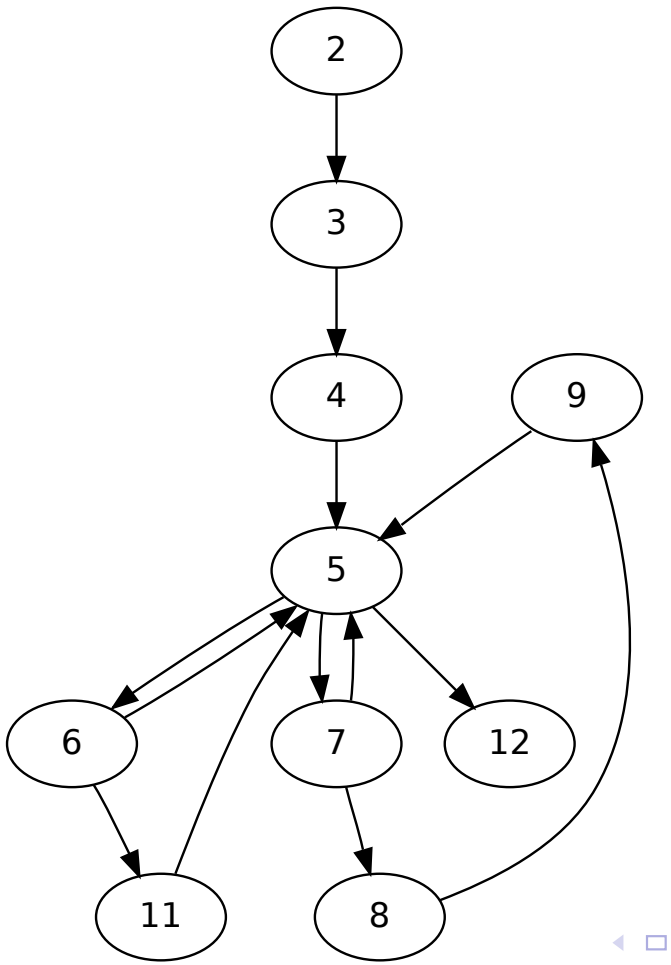
- The sequence of systems calls constitutes a graph that using GraphCreator:
  1. is converted into a graph of ActionNodes,
  2. is exported using Graph Markup Language (GML) format and stored in a database.
- Finally, GraphCreator joins all the files together into a single GML file, to facilitate to process of subgraph mining.

Subgraph Mining

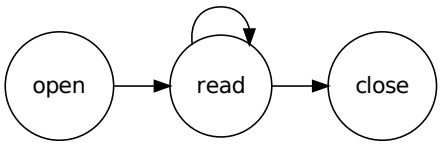
- From the GML containing all the malware multi-graphs, CAMAS SubGraphMiner extracts those subgraphs common across malware.
- SubGraphMiner returns a GML containing a set of subgraphs with:
  - different frequency,
  - node count,
  - edge count.

Subgraph Example

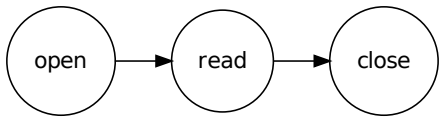
Example of one of the subgraphs found in the GML file: each node identifier (2, 3, 4, etc) refers to an ActionNode identifier.



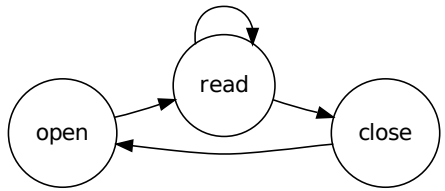
ActionNodes Examples



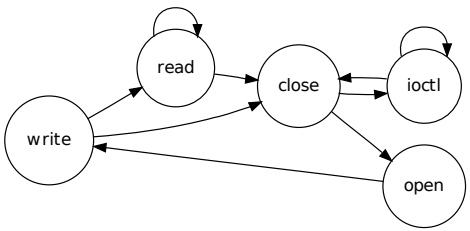
9



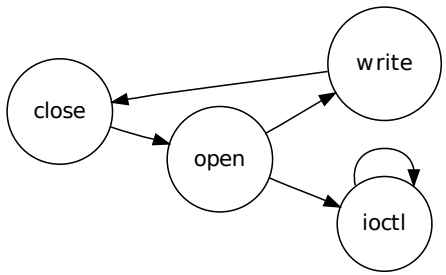
12



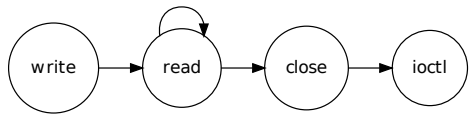
13



19



22



24

Monitoring Apps

- For each new application, several graphs are generated that represent the application usage.
- The union of these graphs is converted into a multi-graph of ActionNodes and stored in a GML file.
- CAMAS SubgraphFinder searches if each of the subgraphs stored in GML subgraph files are found in the app GML.
- SubgraphFinder exploits Bayesian classifiers and artificial neural networks to distinguish a malicious execution from a good one.
- The presence or absence of a subgraph in an app execution is used as feature to discern between a malicious application and a good one.

Testbed

- We have collected data from 13 malicious apps and 7 good ones:
  - AngryBirds, FruitNinja, Google Calendar, Launcher, Android Contact Manager, Calculator, Messages.
- We have collected more than 500 distinct runs, and in the end CAMAS has managed to find 80 distinct vectors.
- From these vectors, there are 33 vectors that are considered as belonging to the malicious class.
- The other 47 vectors extracted from runs of non malicious applications are considered genuine.

Testbed

- We have used this dataset to extract training and testing sets, using the holdout method:
  - selects 70% of the vectors to be used for the training set and the remaining 30% to be used as testing set.
- In the experiments, the following classifiers have been used:
  - linear discriminant classifier (LDC),
  - quadratic discriminant classifier (QDC),
  - k-nearest neighbor (K-NN),
  - artificial neural networks (ANN).

Results

- The K-NN classifier, with a number of neighbors $k = 2$, has produced the best results. Here is the confusion matrix.

	Good	**Malicious**	
Good	12	2	14
Malicious	0	10	10
	12	12	24

- All the malicious vectors have been classified as malicious, whilst 2 good vectors have been wrongly classified as malicious.

Results

Overhead of the whole CAMAS framework:

- tracing overhead: less than 1%;
- graph generation overhead: less than 1 sec;
- graph mining overhead: manually tuned to avoid the generation of files with thousand of subgraphs. When refined, usually it takes 1-2 seconds for each subgraph collection;
- graph searching overhead: order of milliseconds;
- classification overhead: order of milliseconds.

Conclusion and Future Works

- CAMAS, a framework for the analysis and classification of malicious Android apps, through pattern recognition on execution graphs.
- CAMAS finds common subgraphs in malware executions and classifies apps by searching for common patterns of mined subgraphs.
- A future extension considers the detection of high level and security relevant events, such as network actions.
- Increase the number of monitored system calls and explore different relations used to build ActionNodes, e.g. their criticality class.
- Include a larger dataset and different types of classifiers and feature-selection algorithms.

Questions?

- E-mail: daniele.sgandurra@iit.cnr.it